TREK

TELEMETRY PROCESSING

APPLICATION PROGRAMMING INTERFACE (API)

REFERENCE MANUAL

November 2012

TABLE OF CONTENTS

PARAGRAPH PAGE

1 Introduction

The TReK Telemetry Processing (TP) Application Programming Interface (API) was created to provide an easy way to programmatically perform Telemetry Processing functions from various TReK COTS products. The Telemetry Processing API has been designed to be compatible with an ANSI C interface since most COTS products on the market today provide an ANSI C interface. The goal is for the Telemetry Processing API to be compatible with both existing COTS products and any new COTS product on the market which support an ANSI C interface.

Please note that not all of the Telemetry Processing API functions can be used with every TReK COTS product. Some of the TReK COTS products do not support a complete ANSI C interface. This has placed some constraints on which Telemetry Processing API functions are available via the TREK Telemetry Processing API ↔ TReK COTS interface. Section 9 outlines which Telemetry Processing API functions can be used with which TReK COTS products.

2 Related Documents

There are several documents that should be reviewed before reading this document. They are listed in order of priority and are as follows:

TReK Getting Started Guide (TREK-USER-001)
TReK Telemetry Tutorial (TREK-USER-002)
TReK Telemetry Processing Tutorial (TREK-USER-017)
TReK Telemetry Processing User Guide (TREK-USER-003)

It is strongly recommended that you read these documents before you read this document. This document assumes you are familiar with common Telemetry Processing functions such as Adding a Packet and Activating a Packet. Many of the concepts presented in this document will not make sense unless you are familiar with the information in the TReK Telemetry Processing User Guide and Telemetry Processing Tutorial.

3 Telemetry Processing API Implementation

The TReK Telemetry Processing API has been implemented as a Windows Dynamic Linked Library (DLL) which is comprised of multiple C functions. These C functions provide the capability to perform Telemetry Processing function programmatically. The TReK Telemetry Processing API DLL is designed to be thread safe meaning a multithreaded application may safely use the Telemetry Processing API's functions in all of the application's threads.

4 Experience Level Requirements

To use the TReK Telemetry Processing API you must be familiar with the C programming language (an intermediate level C programmer), the Telemetry Processing API functions, and the COTS product you plan to use in conjunction with the Telemetry Processing API. Neither this document nor any other TReK documentation will address how to use the TReK COTS products. Please refer to the documentation that came with the COTS product for that type of information.

5 Detailed Telemetry Processing API Information Available In On-Line Help

This document does not contain all the information available about the TReK Telemetry Processing API. Please reference the Telemetry Processing API Reference Manual On-Line Help for details about Telemetry Processing API function input arguments, output arguments, return codes, and example code.

6 Telemetry Processing API Library, Function Prototypes, and Constants

In order to use the TReK Telemetry Processing API you must have access to the TReK Telemetry Processing API library, function prototypes, and constant definitions. Each TReK COTS product provides access to libraries, function prototypes, and constants in a different way. For example, when you use Visual C++ you will include three header files (trek.h, trek_error.h, trek_tp_user_api.h), that contain the TReK Telemetry Processing API function prototypes and constant definitions. However, when you use Visual Basic, you will include the trek_tp_user_api.bas file.

7 Performance Considerations

The TReK Telemetry Processing API provides programmatic access to Telemetry Processing functions. It is possible to overwhelm the Telemetry Processing application by initiating a large number of requests rapidly one after another. For example, suppose you want to add and activate 20 packets. While it may be possible to rapidly add the packets, activating the packets will be both memory and compute intensive. Therefore, adding and activating all 20 packets rapidly one after another may result in one or more failed activations. We recommend that you use Sleep statements to slow the process down a bit and allow the Telemetry Processing application to finish its work in a moderated fashion. In the case of activating multiple packets, adding a 100 millisecond sleep statement in between each ActivatePacket call may be all it takes to moderate the activation activities. Each situation will be different, but be aware that the following types of functions can be both memory and compute intensive and warrant caution:

ActivatePacket
DeletePacket

8 TReK COTS Products

There are two primary TReK COTS products: Microsoft Visual C++ and Microsoft Visual Basic. Each of these COTS products was selected to meet a particular TReK requirement. Visual C++ was selected to cover the TReK Data Display requirements and the TReK Computation requirements. Visual Basic was selected to cover the TReK Data Display requirements and the TReK Scripting requirements.

Although each COTS product offers an abundant amount of functionality, there are some constraints that were introduced by Visual Basic. For example, Visual Basic does not support the unsigned integer data type. For these reasons, not every API function can be used with each COTS product. Please refer to section 9 for more information.

9 Telemetry Processing API ↔ TReK COTS Product Cross-Reference

This section identifies which Telemetry Processing API functions can be used with which TReK COTS products. Please note that these are the only configurations which have been tested.

Visual Basic
No Visual Basic support at this time.

Visual C++
InitializePacketProperties
InitializeRecordingProperties
AddAPacket
ActivatePacket
UpdatePacketRecordingProperties
StartPacketRecording
StopPacketRecording
GetPacketProperties
GetPacketStatus
GetPacketList
DeletePacket
RefreshTelemetryProcessingMainWindow
GetTPUserAPIReturnCodeAsString

10 Telemetry Processing API Function Descriptions

This section contains the function definition for each function in the Telemetry Processing API. For detailed information on the input parameters, return values and examples see the online help for the Telemetry Processing API.

10.1 InitializePacketProperties

DESCRIPTION

This function initializes a Packet_Properties structure as described below. The calling application is responsible for creating the memory for the Packet_Properties structure. It is a good idea to call this function before populating your Packet_Properties structure with your packet properties data. This ensures that all members of the structure have been initialized. Please note that this data does not define a valid packet since there are several fields that do not contain valid input data (e.g., the packet_id and database members are initialized to empty strings).

Structure Member	Initialized Value
packet_id	Empty String
database	Empty String
packet_type	PDSS_PAYLOAD
data_mode	REAL_TIME
processing	process_entire_packet
packet_source	network
network_protocol	udp
local_ip_address	127.0.0.1
local_port_number	6100
use_join_multicast_flag	join_multicast_off
multicast_address_list.number_addresses	0
remote_server_ip_address	Empty String
remote_server_port_number	0
device_reference	port_name_reference
device_port_name	COM1
device_guid	Empty String
device_api_library	Empty String
packet_rate_mode	pkts_per_sec_mode
expected_pkts_per_sec	1
expected_bits_per_sec	1000

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int InitializePacketProperties(
```

```
Packet_Properties *packet_properties_ptr);
```

Note: The trek_tp_user_api library must be linked into your application.

10.2 InitializeRecordingProperties

DESCRIPTION

This function initializes a Record_Properties structure as described below. The calling application is responsible for creating the memory for the Record_Properties structure. It is a good idea to call this function before populating your Record_Properties structure with your recording properties. This ensures that all members of the structure have been initialized. Please note that this data does not define valid recording properties since there are several fields that do not contain valid input data (e.g., the base_filename and directory members are initialized to empty strings).

Structure Member	Initialized Value
base_filename	Empty String
directory	Empty String
maximum_file_size	10485760
use_max_time_file_is_open_flag	max_time_file_is_open_off
maximum_time_file_is_open	15
use_max_time_directory_is_open_flag	max_time_dir_is_open_off
close_directory_mode	close_directory_in_months_mode

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int InitializeRecordingProperties(
            Record_Properties *record_properties_ptr);
```

Note: The trek_tp_user_api library must be linked into your application.

10.3 AddAPacket

DESCRIPTION

This function sends a request to add a packet. Please review the information in the Structures section for details about setting properties in the Packet_Properties structure.

The `refresh_flag` is an optional input parameter. Valid values are: `refresh_off` and `refresh_on`. If you do not include it, the value will be set to `refresh_on`. Refreshing the Telemetry Processing main window can be a CPU intensive function when there are a large number of packets in the packet list. When this flag is set to refresh_off, it tells the Telemetry Processing application not to refresh the Telemetry Processing main window while performing the AddAPacket function. This is useful if you are calling the AddAPacket function multiple times and only need the main window to refresh after you have completed adding all the packets you wish to add to the packet list. This provides a way to reduce the number of times the main window is updated, thereby increasing performance.

Note: There are other functions that can cause the Telemetry Processing main window to update. So it is possible for the main window to refresh even if you have set this flag to `refresh_off`. This flag only controls the refresh associated with this function's activity.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int AddAPacket(
            Packet_Properties *packet_properties_ptr,
            refresh_type refresh_flag = refresh_on);
```

Note: The trek_tp_user_api library must be linked into your application.

10.4 ActivatePacket

DESCRIPTION

This function sends a request to activate a packet. This function will return SUCCESS if it successfully initiates the activation process. The event corresponding to the event name passed in will be signaled when the activation function completes. The signal does not indicate that the function was successful. You must request a packet status or a packet list to determine whether the function completed successfully. It is valid to leave the `event_name` parameter blank. If you pass in an empty string, the Telemetry Processing application will not create or signal the event.

The `refresh_flag` is an optional input parameter. Valid values are: `refresh_off` and `refresh_on`. If you do not include it, the value will be set to `refresh_on`. Refreshing the Telemetry Processing main window can be a CPU intensive function when there are a large number of packets in the packet list. When this flag is set to refresh_off, it tells the Telemetry Processing application not to refresh the Telemetry Processing main window while performing the ActivatePacket function. This is useful if you are calling the ActivatePacket function multiple times and only need the main window to refresh after all the packets have activated. This provides a way to reduce the number of times the main window is updated, thereby increasing performance.

Note: There are other functions that can cause the Telemetry Processing main window to update. So it is possible for the main window to refresh even if you have set this flag to `refresh_off`. This flag only controls the refresh associated with this function's activity.

Please review the information in the Performance Considerations section of this document for important performance information that should be considered when using this function.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int ActivatePacket(
            char *packet_id,
            int packet_type,
            int data_mode,
            char *event_name,
            refresh_type refresh_flag = refresh_on);
```

Note: The trek_tp_user_api library must be linked into your application.

10.5 UpdatePacketRecordingProperties

DESCRIPTION

This function sends a request to update a packet's recording properties. Packet recording must be off. Please review the information in the Structures section for details about setting properties in the Record_Properties structure.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int UpdatePacketRecordingProperties(
            char *packet_id,
            int packet_type,
            int data_mode,
            Record_Properties *record_properties_ptr);
```

Note: The trek_tp_user_api library must be linked into your application.

10.6 StartPacketRecording

DESCRIPTION

This function sends a request to start packet recording. Packet recording must be off.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int StartPacketRecording(
            char *packet_id,
            int packet_type,
            int data_mode);
```

Note: The trek_tp_user_api library must be linked into your application.

10.7 StopPacketRecording

DESCRIPTION

This function sends a request to stop packet recording. Packet recording must be on.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int StopPacketRecording(
            char *packet_id,
            int packet_type,
            int data_mode);
```

Note: The trek_tp_user_api library must be linked into your application.

10.8 GetPacketProperties

DESCRIPTION

This function requests packet properties information for a specific packet.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int GetPacketProperties(
            Packet_Properties *packet_properties_ptr);
```

Note: The trek_tp_user_api library must be linked into your application.

10.9 GetPacketStatus

DESCRIPTION

This function requests status information for a specific packet.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int GetPacketStatus(
            char *packet_id,
            int packet_type,
            int data_mode,
            Packet_Status *packet_status_ptr);
```

Note: The trek_tp_user_api library must be linked into your application.

10.10 GetPacketList

DESCRIPTION

This function requests a copy of the Telemetry Processing packet list.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int GetPacketList(
            Packet_List_Properties *packet_list_ptr);
```

Note: The trek_tp_user_api library must be linked into your application.

10.11 GetTelemetryStatistics

DESCRIPTION

This function requests telemetry statistics data. The function returns all the telemetry statistics data that is displayed in the Telemetry Processing application's Telemetry Processing Statistics dialog.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"


int GetTelemetryStatistics(
            Telemetry_Statistics *telemetry_statistics_ptr);
```

Note: The trek_tp_user_api library must be linked into your application.

10.12 DeletePacket

DESCRIPTION

This function sends a request to delete a packet. This function will return SUCCESS if it successfully initiates the delete process. The event corresponding to the event name passed in will be signaled when the delete function completes. The signal does not indicate that the function was successful. You must request a packet status or a packet list to determine whether the function completed successfully. It is valid to leave the `event_name` parameter blank. If you pass in an empty string, the Telemetry Processing application will not create or signal the event.

Please review the information in the Performance Considerations section of this document for important performance information that should be considered when using this function.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int DeletePacket(
            char *packet_id,
            int packet_type,
            int data_mode,
            char *event_name);
```

Note: The trek_tp_user_api library must be linked into your application.

10.13 RefreshTelemetryProcessingMainWindow

DESCRIPTION

This function sends a request to the Telemetry Processing application to refresh the data in the main window. Refreshing the Telemetry Processing main window can be a CPU intensive function when there are a large number of packets in the packet list. Several Telemetry Processing API functions, such as the AddAPacket function, contain an input argument called "refresh_flag". When this flag is set to refresh_off, it tells the Telemetry Processing application not to refresh the Telemetry Processing main window while performing the requested function. This is useful if you are calling the AddAPacket function multiple times and only need the main window to refresh after you have completed adding all the packets you wish to add to the packet list. This provides a way to reduce the number of times the main window is updated, thereby increasing performance.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int RefreshTelemetryProcessingMainWindow();
```

Note: The trek_tp_user_api library must be linked into your application.

10.14 GetTPUserAPIReturnCodeAsString

DESCRIPTION

This function provides a way to retrieve a string value that corresponds to a Telemetry Processing API integer value return code.

SYNOPSIS

```
#include "trek.h"
#include "trek_error.h"
#include "trek_tp_user_api.h"

int GetTPUserAPIReturnCodeAsString(
            int code,
            long size_of_data,
            char *message );
```

Note: The trek_tp_user_api library must be linked into your application.

11 Telemetry Processing API Parameters

Parameters	Data Type	Description
packet_properties_ptr	Packet_Properties *	A structure defined in trek.h. You must allocate memory for this variable and populate the appropriate structure members. (Input)
packet_id	char *	The Packet ID (usually the APID). (Input)
packet_type	int	The packet type. Valid values are: CCSDS, CDP, EXPRESS, FDP, GSE, GSE Merge, IMAQ ASCII, PDSS Core, PDSS Payload, PDSS RPSM, PDSS UDSM, PRCU, Suitcase Simulator, UFO, VCDU. (Input)
data_mode	int	The data mode. Valid values are: REAL_TIME, DUMP1, DUMP2, DUMP3, PLAYBACK1, PLAYBACK2, PLAYBACK3, PLAYBACK4, PLAYBACK5, PLAYBACK6, PLAYBACK7, PLAYBACK8, PLAYBACK9, PLAYBACK10, PLAYBACK11, NONE. (Input)
event_name	char *	The name of the event that should be signaled when the requested function completes. It is valid to leave this variable blank. If you pass in an empty string, the Telemetry Processing application will not create or signal the event. (Input)
refresh_flag	refresh_type	Refresh flag. Valid values are: refresh_off, refresh_on. (Input)
record_properties_ptr	Record_Properties *	A structure defined in trek.h. You must allocate memory for this variable and populate the appropriate structure members. (Input)
packet_status_ptr	Packet_Status *	A structure defined in trek.h. You must allocate memory for this variable. (Output)
packet_list_ptr	Packet_List_Properties *	A structure defined in trek.h. You must allocate memory for this variable. (Output)
telemetry_statistics_ptr	Telemetry_Statistics *telemetry_statistics_ptr	A structure defined in trek.h. You must allocate memory for this variable. (Output)
code	unsigned long	A Telemetry Processing API return code. (Input)
size_of_data_ptr	long *	The size of data allocated by the calling program. You must specify enough space for the data to be copied. (Input)

message	char *	Message string containing a description of the return code as a text string. (Output)

12 Telemetry Processing API Structures

This section provides information on populating structures that are passed in to Telemetry Processing API library functions. In some cases these structures contain members that indicate whether data in other structure members should be used. For example, in the Packet_Properties structure there is a member called "use_join_multicast_flag". This member is used to specify whether the "Join Multicast Address Groups" capability is used. If this member is set to "join_multicast_off" then the Telemetry Processing application will ignore the data in the multicast_address_list member. If this member is set to "join_multicast_on" then the Telemetry Processing application will expect the multicast_address_list member to contain one or more multicast address groups that should be joined. The information in the tables below describes the purpose of each structure member and whether that member is always used (it's a required field) or if its use depends on the value of another member (and therefore may be ignored). For more information about how this data is used by the Telemetry Processing application, please reference the Telemetry Processing User Guide.

Packet_Properties Structure

The Packet_Properties structure is used to pass in packet properties.

Structure Member	Description
packet_id	The Packet ID (or APID). The packet_id identified here must be in the database identified in the database member. (Required)
packet_type	The Packet Type. Valid values are: CCSDS, CDP, EXPRESS, FDP, GSE, GSE Merge, IMAQ ASCII, PDSS Core, PDSS Payload, PDSS RPSM, PDSS UDSM, PRCU, Suitcase Simulator, UFO, VCDU. (Required)
data_mode	The Data Mode. Valid values are: REAL_TIME, DUMP1, DUMP2, DUMP3, PLAYBACK1, PLAYBACK2, PLAYBACK3, PLAYBACK4, PLAYBACK5, PLAYBACK6, PLAYBACK7, PLAYBACK8, PLAYBACK9, PLAYBACK10, PLAYBACK11, NONE. (Required)
database	The Database. This must be a complete path. The packet_id identified must be in the database identified by this member. (Required).
processing	The processing to apply to the packets received. Valid values are: off, pass_thru, process_entire_packet, process_on_request_hybrid, process_on_request_only, process_selected_parameters. (Required)

Structure Member	Description
packet_source	Indicates whether the data will arrive via a network interface or a device interface. Valid values are: network, device. (Required)
network_protocol	The network protocol to use when receiving packets via a network interface. Valid values are: udp, tcp_server, tcp_client. This field is required if the packet_source member is set to network. Otherwise this field is ignored.
local_ip_address	The local IP address to be used when receiving packets via a network interface. This must be a valid unicast IP address (loopback address is considered valid). This field is required if the packet_source member is set to network. Otherwise this field is ignored.
local_port_number	The local port number to be used when receiving packets via a network interface. This field is required if the packet_source member is set to network. Otherwise this field is ignored.
use_join_multicast_flag	Indicates whether the Join Multicast Address Groups feature should be used. Valid values are: join_multicast_off, join_multicast_on. This field is required if the packet_source member is set to network and the network_protocol member is set to udp. Otherwise this field is ignored.
multicast_address_list	The multicast address list information. This field is required if the packet_source member is set to network and the use_join_multicast_flag member is set to join_multicast_on. Otherwise this field is ignored.
remote_server_ip_address	The remote server IP address. This must be a valid unicast address. This field is required if the packet_source member is set to network and the network_protocol member is set to tcp_client. Otherwise this field is ignored.
remote_server_port_number	The remote server port number. This field is required if the packet_source member is set to network and the network_protocol member is set to tcp_client. Otherwise this field is ignored.
device_reference	Indicates whether the device identifier provided is a port_name, a GUID, or an API Library. Valid values are: port_name_reference, guid_reference, or api_library_reference. This field is required if the packet_source member is set to device. Otherwise this field is ignored.
device_port_name	The device port name. This field is required if the packet_source member is set to device and the device_reference member is

Structure Member	Description
	set to `port_name_reference`. Otherwise this field is ignored.
`device_guid`	The device GUID. This field is required if the `packet_source` member is set to `device` and the `device_reference` member is set to `guid_reference`. Otherwise this field is ignored.
`device_api_library`	The device API Library. This field is required if the `packet_source` member is set to `device` and the `device_reference` member is set to `api_library_reference`. Otherwise this field is ignored.
`packet_rate_mode`	Indicates whether the packet rate provided is in packets per second or bits per second. Valid values are: `pkts_per_sec_mode`, `bits_per_sec_mode`. (Required)
`expected_pkts_per_sec`	The expected packet rate expressed in packets per second. This field is required if the `packet_rate_mode` member is set to `pkts_per_sec_mode`. Otherwise this field is ignored.
`expected_bits_per_sec`	The expected packet rate expressed in bits per second. This field is required if the `packet_rate_mode` member is set to `bits_per_sec_mode`. Otherwise this field is ignored.

Record_Properties Structure

The Record_Properties structure is used to pass in recording properties.

Structure Member	Description
`base_filename`	The base filename for the recording files. (Required)
`directory`	The directory information for the recording files. This must be a complete path. (Required)
`maximum_file_size`	The maximum file size for the recording files. This value must be greater than or equal to `10240` and less than or equal to `1048576000`. (Required)
`use_max_time_file_is_open_flag`	Indicates whether the Maximum Time File Is Open feature should be used. Valid values are: `max_time_file_is_open_off`, `max_time_file_is_open_on`. (Required)
`maximum_time_file_is_open`	The maximum time (in minutes) the recording files should be open. The maximum time file is open value must be greater than or equal to `1` and less than `71582`. This field is required if the

Structure Member	Description
	use_max_time_file_is_open_flag member is set to max_time_file_is_open_on. Otherwise this field is ignored.
use_max_time_directory_is_open_flag	Indicates whether the Maximum Time Directory Is Open feature should be used. Valid values are: max_time_dir_is_open_off, max_time_dir_is_open_on. (Required)
close_directory_mode	The close directory mode. Valid values are: close_directory_in_days_mode, close_directory_in_weeks_mode, close_directory_in_months_mode, close_directory_in_years_mode. This field is required if the use_max_time_directory_is_open_flag member is set to max_time_dir_is_open_on. Otherwise this field is ignored.

13 Telemetry Processing API Return Codes

Integer Return Value	Return Code Value	Description
0	SUCCESS	The requested function completed successfully
1	FAIL	The requested function failed and a specific error could not be determined.
44001	TP_USER_API_INVALID_PACKET_ID	This error indicates that the packet id value passed in was invalid. The packet id must be in the database identified.
44002	TP_USER_API_INVALID_PACKET_TYPE	This error indicates that the packet type value passed in was invalid. Valid values are: CCSDS, CDP, EXPRESS, FDP, GSE, GSE Merge, IMAQ ASCII, PDSS Core, PDSS Payload, PDSS RPSM, PDSS UDSM, PRCU, Suitcase Simulator, UFO, VCDU.
44003	TP_USER_API_INVALID_DATA_MODE	This error indicates that the data mode value passed in was invalid. Valid values are: REAL_TIME, DUMP1, DUMP2, DUMP3, PLAYBACK1, PLAYBACK2, PLAYBACK3, PLAYBACK4, PLAYBACK5, PLAYBACK6, PLAYBACK7, PLAYBACK8, PLAYBACK9, PLAYBACK10, PLAYBACK11, NONE.
44004	TP_USER_API_INVALID_PROCESSING	This error indicates that the processing value passed in was invalid. Valid values are: off, pass_thru, process_entire_packet, process_on_request_hybrid,

Integer Return Value	Return Code Value	Description
		process_on_request_only, process_selected_parameters.
44005	TP_USER_API_PACKET_NOT_FOUND	This error indicates that the packet identified cannot be found.
44006	TP_USER_API_DUPLICATE_PACKET	This error indicates that the packet to be added already exists.
44007	TP_USER_API_PACKET_ALREADY_ACTIVE	This error indicates that the packet to be activated is already active.
44008	TP_USER_API_INVALID_DATABASE	This error indicates that the database value passed in was invalid.
44009	TP_USER_API_INVALID_PACKET_SOURCE	This error indicates that the packet source value passed in was invalid. Valid values are: network, device.
44010	TP_USER_API_INVALID_NETWORK_PROTOCOL	This error indicates that the network protocol value passed in was invalid. Valid values are: udp, tcp_server, tcp_client.
44011	TP_USER_API_INVALID_LOCAL_IP_ADDRESS	This error indicates that the local IP Address value passed in was invalid.
44012	TP_USER_API_INVALID_LOCAL_PORT_NUMBER	This error indicates that the local port number value passed in was invalid.
44013	TP_USER_API_INVALID_USE_JOIN_MULTICAST_FLAG	This error indicates that the use join multicast flag value passed in was invalid. Valid values are: join_multicast_off, join_multicast_on.
44014	TP_USER_API_INVALID_MULTICAST_ADDRESS_LIST	This error indicates that the multicast address list information passed in was invalid. If you are joining one or more multicast address lists, be sure that the number of addresses value matches the number of addresses provided. Also check to be sure all addresses provided are valid multicast addresses.
44015	TP_USER_API_INVALID_PACKET_RATE_MODE	This error indicates that the packet rate mode value you passed in was invalid. Valid values are: pkts_per_sec_mode, bits_per_sec_mode.
44016	TP_USER_API_INVALID_EXPECTED_PACKETS_PER_SECOND_RATE	This error indicates that expected_pkts_per_sec value you passed in was invalid. The expected packets per second value must be greater than 0.
44017	TP_USER_API_INVALID_EXPECTED_BITS_PER_SECOND_RATE	This error indicates that expected_bits_per_sec value you passed in was invalid. The expected bits per second value must be greater than 0.
44018	TP_USER_API_PACKET_TYPE_REQUIRES_NONE_DATA_M	This error indicates that the packet type passed in must be

Integer Return Value	Return Code Value	Description
	ODE	used with a `None` data mode and the data mode passed in was not `None`.
44019	TP_USER_API_INVALID_USE_OF_NONE_DATA_MODE	This error indicates that the data mode passed in was `None`, but it is not legal to use the `None` data mode with the packet type passed in.
44020	TP_USER_API_PORT_NUMBER_PACKET_TYPE_CONFLICT	This error indicates that the port number you specified is in conflict with the ports already in use by other packets. For example, Suitcase Simulator packets must be sent to a different port than PRCU packets. If an existing Suitcase Simulator packet is already using port 6100, then you cannot add a PRCU packet with a port of 6100.
44021	TP_USER_API_PACKET_IN_USE	This error will occur if you attempt to update a packet while another user is updating the packet. Other users include the Telemetry Processing User Interface user and other programmatic API users.
44022	TP_USER_API_INVALID_REMOTE_SERVER_IP_ADDRESS	This error indicates that the remote server IP address you passed in was invalid.
44023	TP_USER_API_INVALID_REMOTE_SERVER_PORT_NUMBER	This error indicates that the remote server port number you passed in was invalid.
44024	TP_USER_API_INVALID_DEVICE_REFERENCE	This error indicates that the device reference value you passed in was invalid. Valid values are: `port_name_reference, guid_reference`.
44025	TP_USER_API_INVALID_DEVICE_PORT_NAME	This error indicates that the device port name you passed in was invalid.
44026	TP_USER_API_INVALID_DEVICE_GUID	This error indicates that the device GUID you passed in was invalid.
44027	TP_USER_API_INVALID_DEVICE_API_LIBRARY	This error indicates that the device API Library you passed in was invalid.
44028	TP_USER_API_RECORDING_MUST_BE_OFF_DURING_UPDATE	This error will occur if you attempt to update recording properties while recording is on.
44029	TP_USER_API_RECORDING_DIRECTORY_INVALID	This error indicates that you have passed in a recording directory that is invalid.
44030	TP_USER_API_RECORDING_BASE_FILENAME_INVALID	This error indicates that you have passed in a recording base filename that is invalid. This could be the result of using an illegal character in the base filename. The following characters are considered illegal: `\ / : * ? < > |`
44031	TP_USER_API_INVALID_USE_MAX_TIME_FILE_IS_OPEN_F	This error indicates that the

Integer Return Value	Return Code Value	Description
	LAG	`use_max_time_file_is_open_flag` value you passed in was invalid. Valid values are: `max_time_dir_is_open_off`, `max_time_dir_is_open_on`.
44032	TP_USER_API_INVALID_USE_MAX_TIME_DIR_IS_OPEN_FLAG	This error indicates that the `use_max_time_directory_is_open_flag` value you passed in was invalid. Valid values are: `max_time_file_is_open_off`, `max_time_file_is_open_on`.
44033	TP_USER_API_INVALID_CLOSE_DIRECTORY_MODE	This error indicates that the close directory mode value you passed in was invalid. Valid values are: `close_directory_in_days_mode`, `close_directory_in_weeks_mode`, `close_directory_in_months_mode`, `close_directory_in_years_mode`.
44034	TP_USER_API_RECORDING_MAXIMUM_FILE_SIZE_INVALID	This error indicates that the maximum file size you passed in was invalid. The maximum file size must be greater than or equal to `10240` and less than or equal to `1048576000`.
44035	TP_USER_API_RECORDING_MAXIMUM_TIME_FILE_IS_OPEN_INVALID	This error indicates that the maximum time file is open value you passed in was invalid. The maximum time file is open value must be greater than or equal to `1` and less than `71582`.
44036	TP_USER_API_RECORDING_ALREADY_STARTED	This error occurs if you attempt to start packet recording and recording is already in progress.
44037	TP_USER_API_RECORDING_ALREADY_STOPPED	This error occurs if you attempt to stop packet recording and recording is in an off or stopped state.
44038	TP_USER_API_PACKET_MUST_BE_ACTIVE	This error occurs if you attempt to perform a function that can only be performed when a packet is active and the packet is currently inactive
44039	TP_USER_API_PACKET_SHUTDOWN_IN_PROGRESS	This error occurs if you attempt to perform a function on a packet that is currently being shutdown (deactivated, deleted, etc.).
44040	TP_USER_API_RECONFIGURATION_IN_PROGRESS	This error occurs if you attempt to perform a function while the Telemetry Processing application is in the middle of a reconfiguration (the Telemetry Processing graphical user interface user has initiated a New, Open, etc.).
44041	TP_USER_API_PACKET_ACTIVATION_IN_PROGRESS	This error indicates that the packet you are attempting to modify is being activated.

Integer Return Value	Return Code Value	Description
44042	TP_USER_API_INVALID_REFRESH_FLAG	This error indicates that the `refresh_flag` value you passed in was invalid. Valid values are: `refresh_off`, `refresh_on`.
44043	TP_USER_API_TIMEOUT_WHILE_WAITING_TO_ACCESS_DATABASE	This error indicates that a timeout occurred while waiting to access the database.
44044	TP_USER_API_NOT_ENOUGH_SPACE	This error indicates that you have not allocated enough space to hold the information that needs to be returned.
44045	TP_USER_API_INVALID_API_RETURN_CODE	This error will occur if you pass an invalid TP User API return code to the `GetTPUserAPIReturnCodeAsString` function.
44046	TP_USER_API_NO_STATISTICS_DATA_AVAILABLE	This error indicates that there is no statistics data available.
44047	TP_USER_API_ERROR_ACCESSING_STATISTICS_DATA	This error indicates that an error occurred while attempting to access a file that contains the statistics data being requested.
44048	TP_USER_API_TIMEOUT_ERROR	This error indicates that a timeout occurred when the Telemetry Processing API library tried to communicate with the Telemetry Processing application.

Appendix A Acronyms

Note: This acronym list is global to all TReK documentation. Some acronyms listed may not be referenced within this document.

AOS	Acquisition of Signal
API	Application Programming Interface
APID	Application Process Identifier
ASCII	American Standard Code for Information Interchange
CAR	Command Acceptance Response
CAR1	First Command Acceptance Response
CAR2	Second Command Acceptance Response
CCSDS	Consultative Committee for Space Data Systems
CDB	Command Database
CDP	Custom Data Packet
COR	Communication Outage Recorder
COTS	Commercial-off-the-shelf
CRR	Command Reaction Response
DSM	Data Storage Manager
EHS	Enhanced Huntsville Operations Support Center (HOSC)
ERIS	EHS Remote Interface System
ERR	EHS Receipt Response
EXPRESS	Expediting the Process of Experiments to the Space Station
ES	Expected State
FAQ	Frequently Asked Question
FDP	Functionally Distributed Processor
FSV	Flight System Verifier
FSV1	First Flight System Verifier
FSV2	Second Flight System Verifier
FPD	Flight Projects Directorate
FTP	File Transfer Protocol
GMT	Greenwich Mean Time
GRT	Ground Receipt Time
GSE	Ground Support Equipment
HOSC	Huntsville Operations Support Center
ICD	Interface Control Document
IMAQ ASCII	Image Acquisition ASCII
IP	Internet Protocol
ISS	International Space Station
LDP	Logical Data Path
LES	Limit/Expected State
LOR	Line Outage Recorder
LOS	Loss of Signal
MCC-H	Mission Control Center – Houston
MOP	Mission, Operational Support Mode, and Project
MSFC	Marshall Space Flight Center
MSID	Measurement Stimulus Identifier

NASA	National Aeronautics and Space Administration
OCDB	Operational Command Database
OS	Operating System
PC	Personal Computer, also Polynomial Coefficient
PCDB	POIC Project Command Database
PDL	Payload Data Library
PDSS	Payload Data Services System
PGUIDD	POIC to Generic User Interface Definition Document
POIC	Payload Operations Integration Center
PP	Point Pair
PRCU	Payload Rack Checkout Unit
PSIV	Payload Software Integration and Verification
RPSM	Retrieval Processing Summary Message
SC	State Code
SCS	Suitcase Simulator
SSP	Space Station Program
SSCC	Space Station Control Center
SSPF	Space Station Processing Facility
TCP	Transmission Control Protocol
TReK	Telescience Resource Kit
TRR	TReK Receipt Response
TSC	Telescience Support Center
UDP	User Datagram Protocol
UDSM	User Data Summary Message
URL	Uniform Resource Locator
USOS	United States On-Orbit Segment
VCDU	Virtual Channel Data Unit
VCR	Video Cassette Recorder
VPN	Virtual Private Network

Appendix B Glossary

Note: This Glossary is global to all TReK documentation. All entries listed may not be referenced within this document.

Application Programming Interface (API)	A set of functions used by an application program to provide access to a system's capabilities.
Application Process Identifier (APID)	An 11-bit field in the CCSDS primary packet header that identifies the source-destination pair for ISS packets. The type bit in the primary header tells you whether the APID is a payload or system source-destination.
Calibration	The transformation of a parameter to a desired physical unit or text state code.
Communications Outage Recorder	System that captures and stores payload science, health and status, and ancillary data during TDRSS zone of exclusion.
Consultative Committee for Space Data Systems (CCSDS) format	Data formatted in accordance with recommendations or standards of the CCSDS.
Consultative Committee for Space Data Systems (CCSDS) packet	A source packet comprised of a 6-octet CCSDS defined primary header followed by an optional secondary header and source data, which together may not exceed 65535 octets.
Conversion	Transformation of downlinked spacecraft data types to ground system platform data types.
Custom Data Packet	A packet containing a subset of parameters that can be selected by the user at the time of request.
Cyclic Display Update Mode	A continuous update of parameters for a particular display.
Decommutation (Decom)	Extraction of a parameter from telemetry.
Discrete Values	Telemetry values that have states (e.g., on or off).

Dump	During periods when communications with the spacecraft are unavailable, data is recorded onboard and played back during the next period when communications resume. This data, as it is being recorded onboard, is encoded with an onboard embedded time and is referred to as dump data.
Enhanced HOSC System (EHS)	Upgraded support capabilities of the HOSC systems to provide multi-functional support for multiple projects. It incorporates all systems required to perform data acquisition and distribution, telemetry processing, command services, database services, mission support services, and system monitor and control services.
Exception Monitoring	A background process capable of continuously monitoring selected parameters for Limit or Expected State violations. Violation notification is provided through a text message.
Expected State Sensing	Process of detecting a text state code generator in an off-nominal state.
EXPRESS	An EXPRESS Rack is a standardized payload rack system that transports, stores and supports experiments aboard the International Space Station. EXPRESS stands for EXpedite the PRocessing of Experiments to the Space Station.
File transfer protocol (ftp)	Protocol to deliver file-structured information from one host to another.
Flight ancillary data	A set of selected core system data and payload health and status data collected by the USOS Payload MDM, used by experimenters to interpret payload experiment results.

Grayed out	Refers to a menu item that has been made insensitive, which is visually shown by making the menu text gray rather than black. Items that are grayed out are not currently available.
Greenwich Mean Time (GMT)	The solar time for the meridian passing through Greenwich, England. It is used as a basis for calculating time throughout most of the world.
Ground ancillary data	A set of selected core system data and payload health and status data collected by the POIC, which is used by experimenters to interpret payload experiment results. Ground Ancillary Data can also contain computed parameters (pseudos).
Ground receipt time	Time of packet origination. The time from the IRIG-B time signal received.
Ground Support Equipment (GSE)	GSE refers to equipment that is brought in by the user (i.e. equipment that is not provided by the POIC).
Ground Support Equipment Packet	A CCSDS Packet that contains data extracted from any of the data processed by the Supporting Facility and the format of the packet is defined in the Supporting Facility's telemetry database.
Huntsville Operations Support Center (HOSC)	A facility located at the Marshall Space Flight Center (MSFC) that provides scientists and engineers the tools necessary for monitoring, commanding, and controlling various elements of space vehicle, payload, and science experiments. Support consists of real-time operations planning and analysis, inter- and intra-center ground operations coordination, facility and data system resource planning and scheduling, data systems monitor and control operations, and data flow coordination.

IMAQ ASCII	A packet type that was added to TReK to support a very specific application related to NASA's Return to Flight activities. It is not applicable to ISS. It is used to interface with an infrared camera that communicates via ASCII data.
Limit Sensing	Process of detecting caution and warning conditions for a parameter with a numerical value.
Line Outage Recorder Playback	A capability provided by White Sands Complex (WSC) to play back tapes generated at WSC during ground system communication outages.
Measurement Stimulus Identifier (MSID)	Equivalent to a parameter.
Monitoring	A parameter value is checked for sensing violations. A message is generated if the value is out of limits or out of an expected state.
Parameter	TReK uses the generic term parameter to mean any piece of data within a packet. Sometimes called a measurement or MSID in POIC terminology.
Payload Data Library (PDL)	An application that provides the interface for the user to specify which capabilities and requirements are needed to command and control his payload.
Payload Data Services Systems (PDSS)	The data distribution system for ISS. Able to route data based upon user to any of a number of destinations.
Payload Health and Status Data	Information originating at a payload that reveals the payload's operational condition, resource usage, and its safety/anomaly conditions that could result in damage to the payload, its environment or the crew.
Payload Operations Integration Center (POIC)	Manages the execution of on-orbit ISS payloads and payload support systems in coordination/unison with distributed International Partner Payload Control Centers, Telescience Support Centers (TSC's) and payload-unique remote facilities.

Payload Rack Checkout Unit (PRCU)	The Payload Rack Checkout Unit is used to verify payload to International Space Station interfaces for U.S. Payloads.
Playback	Data retrieved from some recording medium and transmitted to one or more users.
Pseudo Telemetry (pseudo data)	Values that are created from calculations instead of directly transported telemetry data. This pseudo data can be created from computations or scripts and can be displayed on the local PC.
Remotely Generated Command	A command sent by a remote user whose content is in a raw bit pattern format. The commands differ from predefined or modifiable commands in that the content is not stored in the POIC Project Command Database (PCDB).
Science data	Sensor or computational data generated by payloads for the purpose of conducting scientific experiments.
Subset	A collection of parameters from the total parameter set that is bounded as an integer number of octets but does not constitute the packet itself. A mini-packet.
Super sampled	A parameter is super sampled if it occurs more than once in a packet.
Swap Type	A flag in the Parameter Table of the TReK database that indicates if the specified datatype is byte swapped (B), word swapped (W), byte and word swapped (X), byte reversal (R), word reversal (V) or has no swapping (N).
Switching	A parameter's value can be used to switch between different calibration and sensing sets. There are two types of switching on TReK: range and state code.

Transmission Control Protocol (TCP)	TCP is a connection-oriented protocol that guarantees delivery of data.
Transmission Control Protocol (TCP) Client	A TCP Client initiates the TCP connection to connect to the other party.
Transmission Control Protocol (TCP) Server	A TCP Server waits for (and accepts connections from) the other party.
Telemetry	Transmission of data collected form a source in space to a ground support facility. Telemetry is downlink only.
Telescience Support Center (TSC)	A TSC is a NASA funded facility that provides the capability to plan and operate on-orbit facility class payloads and experiments, other payloads and experiments, and instruments.
User Application	Any end-user developed software program that uses the TReK Application Programming Interface software. Used synonymously with User Product.
User Data Summary Message (UDSM)	Packet type sent by PDSS that contains information on the number of packets sent during a given time frame for a PDSS Payload packet. For details on UDSM packets, see the POIC to Generic User IDD (SSP-50305).
Uplink format	The bit pattern of the command or file uplinked.
User Datagram Protocol (UDP)	UDP is a connection-less oriented protocol that does not guarantee delivery of data. In the TCP/IP protocol suite, the UDP provides the primary mechanism that application programs use to send datagrams to other application programs. In addition to the data sent, each UDP message contains both a destination port number and a fully qualified source and destination addresses making it possible for the UDP software on the destination to deliver the message to the correct recipient process and for the recipient process to send a reply.

User Product	Any end-user developed software program that uses the TReK Application Programming Interface software. Used synonymously with User Application.
Web	Term used to indicate access via HTTP protocol; also referred to as the World Wide Web (WWW).